Electronic Notes in Theoretical Computer Science 70 No. 1 (2003)
URL: http://www.elsevier.nl/locate/entcs/volume70.html 14 pages

Feasible functionals and intersection of ramified types

Daniel Leivant

Computer Science Department, Indiana University, Bloomington, IN 47405

Abstract

We show that the basic feasible functions of Cook and Urquhart's **BFF** [8,9] are precisely the functionals definable in a natural system of ramified recurrence that uses type intersection (for tier-variants of a common type). This further confirms the stability of **BFF** as a notion of computational feasibility in higher type. It also suggests the potential usefulness of type-intersection restricted to sort-variants of a common type.

INTRODUCTION: COMPUTABILITY AND FEASIBILITY IN HIGHER TYPE

Computable higher type functionals have been studied for about a century, for several intertwined reasons. One of the first to explicitly consider *feasibility* of functionals was Robert Constable, who in [6] introduced a machine model for functionals, and considered the definability of the functionals computable therein in a certain function algebra.[3] Melhorn [18] refined Constable's algebraic approach by lifting to second order types the characterization given by Cobham [5] of the class FP of functions computable in polynomial time. A corresponding machine model was defined by Kapron and Cook in [13], and shown to be equivalent to Mehlhorn's class.

Another thread in the evolution of the subject was concerned with functional interpretation of proofs in Buss's Bounded Arithmetic. In [2] Buss introduced a system IS^1_2 of arithmetic and showed that its definable functions form precisely FP. In [3] Buss considered the intuitionistic variant of IS^1_2, and

[1] Research partially supported by NSF grant CCR-0105651.
[2] Email: leivant@cs.indiana.edu
[3] See [4] for a correction.

defined a complex functional interpretation which yields a poly-time instantiation theorem for the system. This approach was substantially refined and simplified by Cook and Urquhart in [8,9], where they defined a system **BFF** (for *Basic Feasible Functionals*), based on the typed lambda calculus, and which supports a functional interpretation of IS_1^2, analogous to Gödel's functional interpretation of first order arithmetic [10]. [4] In [14] Cook and Kapron showed that the second order fragment $\mathbf{BFF}_2$ of **BFF** contains precisely the functionals defined in Mehlhorn's system, viz. the same as the functionals computable by the machine model of [13].

It is not immediately clear that $\mathbf{BFF}_2$ should be admitted as a canonical delineation of the feasible second order functionals. Indeed, Cook exhibited in [7] a functional L that might be considered feasible, and yet falls outside $\mathbf{BFF}_2$. Cook stated three conditions that any proposed definition of type 2 feasibility must satisfy, and those are in fact satisfied by $\mathbf{BFF}_2$ appropriately augmented with L. However, Seth showed [19] that when two additional and quite natural conditions are imposed, then $\mathbf{BFF}_2$ emerges as the only admissible notion of feasibility for second order functionals. Nonetheless, it is useful to lift doubts about the robustness of $\mathbf{BFF}_2$, and more generally of the class **BFF**, by providing additional natural characterizations, notably ones that are not tied umbilically to explicit resource restrictions, as are all characterizations above.

Frameworks for characterizing computational complexity classes without any reference to resources have been developed over the last dozen odd years, jointly referred to as *Implicit Computational Complexity.* Included are, among others, ramified functional programs, ramified first order proof systems, higher order logics with restricted set-existence, structural restrictions on applicative terms and proofs, and modal and linear type systems and proof systems. Such formalisms are particularly attractive for delineating notions of feasibility in higher type: they are based on concepts that do not refer directly to functions and computations, and consequently they lift seamlessly to higher type computing.

One implicit characterization of **BFF** was proposed in [12], where a ramified imperative programming language of loop programs is presented, dubbed Type 2 Inflationary Tiered Loop Programs (ITLP_2), which computes in type 2 exactly $\mathbf{BFF}_2$. The imperative framework is appealing from an expository viewpoint, as well as for implementations. However, the formalism of [12] is based on a principle of "inflationary tiers": it posits functions that embed lower (i.e. weaker) tiers under a size-bound into higher tiers, that is functions $\underline{lift}_{ij} : \mathbb{W}_i \to \mathbb{W}_j \to \mathbb{W}_i$ for $i > j$ (where $\mathbb{W}$ is the term algebra representing

[4] Initially the system was denoted PV^ω.

$\{0,1\}^*$ and the subscript designates the tier), such that [5]

$$\underline{\mathit{lift}}_{ij}(x,y) = \begin{cases} y \text{ if } |y| \leq |x| \\ \boldsymbol{\varepsilon} \text{ otherwise} \end{cases}$$

Thus, the system intertwines tiers with explicit bounding of resources, not significantly different from the use of Cobham's bounded recurrence, thereby defeating the very rationale of ramification and similar implicit characterizations of computational complexity.

One problem with ramification in higher type is that within a ramified setting one can define functionals which become unfeasible if ramification restrictions are removed. For instance, once we allow function variables of type $W_i \to W_i$ we can define the iteration functional

$$\lambda f^{W_i \to W_i}\, \lambda x^{W_{i+1}}.\, \mathbf{R}\boldsymbol{\varepsilon} f f x$$

(where $\mathbf{R}$ denotes a recursor operator for $\mathbb{W}$), which is not definable in **BFF**, and indeed maps a feasible function that doubles the input size to an unfeasible function of exponential growth. We cannot bypass this issue by insisting that function variables be assigned types of the form $W_i \to W_j$ with $j > i$, since these could be composed with downward-tier coercion functions to yield function arguments of type $W_i \to W_i$. Assigning to function variables types $W_i \to W_j$, with fixed $j < i$, is also undesirable, since that would exclude the perfectly legitimate self composition functional $\lambda f.\lambda x.f(f(x))$. This issue, and similar ones, imply the need for a more flexible system of ramification for higher type functionals. This is not unexpected, since the mechanics of object tiering uses the definability of tier-reduction functions, which make tier intersection unnecessary: the intersection of several tiers is computationally equivalent to the highest of these tiers. In contrast, no such mechanism is available for higher types.

This issue is addressed in [12] by a tier quantification mechanism, allowing types such as $\forall i.W_{i+1} \to W_i$, where W_i is the type of words in tier i. This device permits to lift to types of rank 1 the implicit mechanism of intersection, but does not seem to have any easy extension to higher types. Indeed, even if all function variables are assigned the type above, compound terms denoting unary functions may have other types, for example $f \circ f$ would have type $\forall i.\, W_{i+2} \to W_i$, and unary constructors would have type $\forall i.\, W_i \to W_i$. The composition of these variables would no longer have this type, leading to the impossibility of assigning properly tiered types to higher order functionals.

In this paper we show that finite intersection of tier-variants of a given type will do the job at all ranks. Our advance over [12] is thus in: (a) Avoiding reference to resource bounds ("inflationary tiering"); and (b) Characterizing

[5] In [12] these functions are named *down*, and tiering is reversed, with lower tiers driving computation in higher tiers.

of **BFF** in *all* finite types. It is of interest that both [12] and our present work extend ramified recurrence in the style of [16], rather than [1]. Indeed, the notion of "safe composition" used in the latter goes contrary to the treatment of ramification as a form of sorting, and it is this treatment that permits natural extension of ramified recurrence to higher type.

While we consider here an adaptation of type ramification to higher types, it is also possible to study **BFF** via weak forms of polymorphism. Results along that line can be found in [17]. Significantly, both approaches yield robust characterizations of **BFF** by lifting to higher type natural type-theoretic characterizations of poly-time.

1 FUNCTIONAL PROGRAMS OVER FREE ALGEBRAS

1.1 Primitive recursion over free algebras

Let $\mathbb{A}$ be the free algebra generated from constructors $\mathbf{c}_1 \dots \mathbf{c}_k$ $(k > 0)$, with $\underline{arity}(\mathbf{c}_i) = r_i \geq 0$, and set $r =_{\mathrm{df}} \max(r_i)$. Special cases of this generic definition include the algebra $\mathbb{N}$ of unary numerals, generated from the zero-ary $\mathbf{0}$ and the unary $\mathbf{s}$ (the successor functions), and the algebra $\mathbb{W} \cong \{0,1\}^*$ of binary words, generated from the zero-ary $\boldsymbol{\varepsilon}$ and the unary $\mathbf{0}$ and $\mathbf{1}$.

The schema of *primitive recursion on* $\mathbb{A}$ allows the definition of a function over $\mathbb{A}$ of arity $q+1$ from functions $g_{c_1} \dots g_{c_k}$, where g_{c_i} are of arity $\leq q+2r_i$, by

$$f(\vec{x}, \mathbf{c}_i(a_1 \dots a_{r_i})) = g_{c_i}(\vec{x}, \vec{a}, f(\vec{x}, a_1), \dots f(\vec{x}, a_{r_i}))$$

It is useful to consider the *monotonic* cases of this schema, in which the functions g_{c_i} have no direct access to $\vec{a}$:

$$f(\vec{x}, \mathbf{c}_i(a_1 \dots a_{r_i})) = g_{c_i}(\vec{x}, f(\vec{x}, a_1), \dots f(\vec{x}, a_{r_i}))$$

We dub this simplified form of primitive recursion *recurrence*.[6] Another restricted form of primitive recursion, orthogonal to recurrence, is the *Branching* schema, where the functions g_{c_i} have no access to the previous value of f:

$$f(\vec{x}, \mathbf{c}_i(a_1 \dots a_{r_i})) = g_{c_i}(\vec{x}, \vec{a})$$

Using Branching we obtain the definition-by-cases and destructor functions:

$$\begin{aligned} \underline{cases}(\mathbf{c}_i(\vec{a}), x_1 \dots x_k) &= x_i \\ \underline{dstr}_j(\mathbf{c}_i(\vec{a})) &= a_j \qquad (0 < j \leq r_i) \\ \underline{dstr}_j(\mathbf{c}_i(\vec{a})) &= \mathbf{c}_i(\vec{a}) \ (r_i < j \leq r) \end{aligned}$$

[6] This schema, for $\mathbb{A} = \mathbb{N}$, is also known as *iteration with parameters*, but the phrase "iteration" is inappropriate for $\mathbb{W}$ and other algebras.

For example, for $\mathbb{A} = \mathbb{N}$ the unique destructor is the cut-off predecessor, and $\underline{cases}(x, y, z) =$ if $(x = 0)$ then y else z. It is easy to see that every instance of the Branching schema is reducible to the cases and destructor functions, using composition.

We are particularly interested in recurrence over the algebra $\mathbb{W}$, that is

$$\begin{aligned} f(\vec{x}, \boldsymbol{\varepsilon}) &= g_\epsilon(\vec{x}) \\ f(\vec{x}, \mathbf{0}w) &= g_0(\vec{x}, f(\vec{x}, w)) \\ f(\vec{x}, \mathbf{1}w) &= g_1(\vec{x}, f(\vec{x}, w)) \end{aligned}$$

The simultaneous definition of a vector $\vec{f}$ of functions is similar, referring to given vectors of functions $\vec{g}_\epsilon$, $\vec{g}_0$ and $\vec{g}_1$.

Let $\boldsymbol{\lambda}_1$ be the simply typed lambda calculus with product types, and corresponding pairing $\langle \cdot, \cdot \rangle$ and projection functions π_0, π_1. In addition to β-reductions, we have here the pairing reduction: $\pi_i \langle \mathbf{t}_0, \mathbf{t}_1 \rangle \mapsto \mathbf{t}_i$. We write ι for the base type, and associate $\to$ to the right. When convenient, we write $(\tau_1 \dots \tau_m) \to \sigma$ for $\tau_1 \to \tau_2 \to \dots \to \tau_m \to \sigma$ $(m \geq 0)$.[7] If all τ_i are all one and the same type τ, we write $\tau^m \to \sigma$ for the above.

Let $\boldsymbol{\lambda}_1(\mathbb{W})$ be the following extension of $\boldsymbol{\lambda}_1$. The identifier $\boldsymbol{\varepsilon}$ is admitted as a constant of type ι, and the identifiers $\mathbf{0}$ and $\mathbf{1}$ as constants of type $\iota \to \iota$. In addition, we include constants for the branching and recurrence operations, B and R, both of type $(\iota, \iota \to \iota, \iota \to \iota, \iota) \to \iota$. The reductions of $\boldsymbol{\lambda}_1$ are augmented with reductions for B and R:

$$\begin{aligned} \mathsf{B}\mathbf{t}_\epsilon \mathbf{t}_0 \mathbf{t}_1 \boldsymbol{\varepsilon} &\to \mathbf{t}_\epsilon \\ \mathsf{B}\mathbf{t}_\epsilon \mathbf{t}_0 \mathbf{t}_1 (\mathbf{i}\mathbf{w}) &\to \mathbf{t}_i(\mathbf{w}) \quad i = 0, 1 \\ \\ \mathsf{R}\mathbf{t}_\epsilon \mathbf{t}_0 \mathbf{t}_1 \boldsymbol{\varepsilon} &\to \mathbf{t}_\epsilon \\ \mathsf{R}\mathbf{t}_\epsilon \mathbf{t}_0 \mathbf{t}_1 (\mathbf{i}\mathbf{w}) &\to \mathbf{t}_i(\mathsf{R}\mathbf{t}_\epsilon \mathbf{t}_0 \mathbf{t}_1 \mathbf{w}) \quad i = 0, 1 \end{aligned}$$

It is clear how function definition by recurrence is conveyed in $\boldsymbol{\lambda}_1(\mathbb{W})$: if f is defined from g_ϵ, g_0, and g_1 as above, and g_i is defined by G_i $(i = \epsilon, 0, 1)$, then f is defined by the term

$$F =_{\mathrm{df}} \lambda x_1 \dots x_m, w.\, \mathsf{R}(G_\epsilon \vec{x})(G_0 \vec{x})(G_1 \vec{x}) w$$

The rendition of the branching schema by B is similar.

1.2 Bounded primitive recursion

In his seminal paper [11] Grzegorczyk gave his famous classification of primitive recursive functions, closing each class under the schema of *bounded recursion,* i.e. the schema that admits a function f if the functions g_0, g_s and j

[7] This simulation of type-product is useful in the product-free version of $\boldsymbol{\lambda}_1$.

are admitted, and [8]

$$\begin{aligned} f(\vec{x},0) &= g_0(\vec{x}) \\ f(\vec{x},\mathbf{s}w) &= g_s(\vec{x},w,f(\vec{x},w)) \\ f(\vec{x},w) &< j(\vec{x},w) \end{aligned}$$

Cobham [5] showed that the functions over $\mathbb{N}$ computable (on a Turing machine) in polynomial time can be characterized by admitting initial functions that yield values of size polynomial in the input's size, and then closing under bounded primitive recursion on words. That is, a function over $\mathbb{W}$ is in FP iff it is definable from the constructors of $\mathbb{W}$, and a size multiplication function $u * v =_{\text{df}} 1^{|u|\cdot|v|}\boldsymbol{\varepsilon}$, using explicit definitions and the following schema **BR** of bounded primitive recursion: [9]

$$\begin{aligned} f(\vec{x},\boldsymbol{\varepsilon}) &= g_\epsilon(\vec{x}) \\ f(\vec{x},\mathbf{i}w) &= g_i(\vec{x},f(\vec{x},w)) \quad (i=0,1) \\ |f(\vec{x},w)| &< |j(\vec{x},w)| \end{aligned}$$

We use the following alternative rendition $\mathbf{BR}'$ of bounded primitive recursion:

$$\begin{aligned} f(\vec{x},\boldsymbol{\varepsilon}) &= g_\epsilon(\vec{x}) \\ f(\vec{x},\mathbf{i}w) &= g_i(\vec{x},w,f(\vec{x},w) \restriction J(\vec{x},w)) \qquad (i=0,1) \end{aligned}$$

Here $u \restriction v$ is the truncation of u to the length of v, e.g. $0010\epsilon \restriction 01\epsilon = 00\epsilon$, and $0010\epsilon \restriction 11111\epsilon = 0010\epsilon$.

Lemma **1** *The schema* $\mathbf{BR}'$ *over* $\mathbb{W}$ *is equivalent, modulo linear time simulations, to* **BR**.

Proof. If f is defined from g_ϵ, g_0, g_1 and j by **BR**, then f is defined from g_ϵ, g_0, g_1 and J by $\mathbf{BR}'$, where $J(\vec{x},w) =_{\text{df}} \max[j(\vec{x},\mathbf{0}w), j(\vec{x},\mathbf{1}w)]$. Conversely, if f is defined from g_ϵ, g_0, g_1 and J by $\mathbf{BR}'$, then f is defined from g_ϵ, g_0, g_1 and j by **BR**, where $j(\vec{x},w) =$ if $w = \boldsymbol{\varepsilon}$ then $g_0(\vec{x},w)$ else $J(\vec{x},p(w))$, where p is the predecessor function. ⊣

1.3 Bounded recurrence

When a bounded primitive recursion is monotonic, we dub it *bounded recurrence:*

[8] This "doctrine of size" for function definition is strikingly similar to "doctrine of size" for function definition is strikingly similar to Zermelo's doctrine of size for taming the comprehension principle of naive set theory: the naive admission of set definition by arbitrary description, $\{x \mid P(x)\}$ is replaced by the Separation Schema, which only admits $\{x \in S \mid P(x)\}$, S an already defined set.

[9] The generic statement of **BR** for arbitrary word algebras is similar. Cobham's phrased this schema as "bounded recursion on notations", and insisted on working with natural numbers. This was in accord with the early focus of mathematical logic on number systems, and the exclusive reference to numeric computing in traditional Recursion Theory.

$$f(\vec{x}, \boldsymbol{\varepsilon}) = g_\epsilon(\vec{x})$$
$$f(\vec{x}, \mathbf{i}w) = g_i(\vec{x}, f(\vec{x}, w) \restriction J(\vec{x}, w)) \qquad (i = 0, 1)$$

Proposition **2** *Each instance of bounded primitive recursion over* $\mathbb{W}$ *can be derived from branching and bounded recurrence.*

The proof is similar to the proof of [16, Lemma 4.2], and uses function-definitions by simultaneous recurrence as an auxiliary notion, where simultaneous recurrence is made possible by the presence of pairing. The simulation requires a quadratic increase in computation time. The reference to $\mathbb{W}$ is essential here, e.g. the argument does not work for $\mathbb{N}$.

In order to incorporate bounded recurrence into a definition of higher type functionals, Cook and Urquhart [8,9] rephrased bounded recurrence as a functional operator, with reduction rules, to be adjoined to the simply typed lambda calculus $\boldsymbol{\lambda}_1$. They dub their system PV^ω; we use here a slight variant of that calculus. Let $\overline{\mathbf{R}}$ be a function identifier of type $\iota \to (\iota \to \iota)^3 \to \iota \to \iota$. The reductions conveying the intended meaning of $\overline{\mathbf{R}}$ are:

$$\overline{\mathbf{R}} G_\epsilon G_0 G_1 J \boldsymbol{\varepsilon} \mapsto G_\epsilon$$
$$\overline{\mathbf{R}} G_\epsilon G_0 G_1 J(\mathbf{i}X) \mapsto (G_i H) \restriction (JX) \qquad (i = 0, 1)$$
$$\text{where } H =_{\mathrm{df}} \overline{\mathbf{R}} G_\epsilon G_0 G_1 JX$$

Thus, if f is defined by bounded recurrence as above, and functions g_ϵ, g_0, g_1 and j are defined by terms G_ϵ, G_0, G_1, and J, then f is defined by the term

$$F =_{\mathrm{df}} \lambda \vec{x}, w.\, \overline{\mathbf{R}}(G_\epsilon \vec{x})(G_0 \vec{x})(G_1 \vec{x})(J \vec{x})\, w$$

Our variant $\boldsymbol{\lambda}_1^-(\mathbb{W})$ of PV^ω is an extension of $\boldsymbol{\lambda}_1$, identical to $\boldsymbol{\lambda}_1(\mathbb{W})$ except for two modifications: (a) The constant $\mathbf{R}$ and the associated reduction rules are replaced by $\overline{\mathbf{R}}$ and its associated reduction rules; (b) The constant $\restriction$ is included as a primitive, with reduction rules $x \restriction \boldsymbol{\varepsilon} \mapsto \boldsymbol{\varepsilon}$, $\boldsymbol{\varepsilon} \restriction x \mapsto \boldsymbol{\varepsilon}$, and $\mathbf{i}x \restriction \mathbf{j}w \mapsto \mathbf{i}(x \restriction w)$.

From Proposition 2 we obtain:

Proposition **3** *A functional over* $\mathbb{W}$ *is definable in* PV^ω *iff it is definable in* $\boldsymbol{\lambda}_1^-(\mathbb{W})$.

2 RAMIFIED RECURRENCE IN FINITE TYPES

2.1 Ramified recurrence

The schema of recurrence over $\mathbb{N}$ embodies an impredicative reading of the natural numbers (and similarly for other algebras $\mathbb{A}$). Consider the definition of exponentiation from the doubling function: doubling is defined by $\mathbf{R0s}^2$, where $\mathbf{s}^2 \equiv \mathbf{s} \circ \mathbf{s} \equiv \lambda z.\, \mathbf{s}(\mathbf{s}(z))$. So base-2 exponentiation is defined by $E =_{\mathrm{df}} \mathbf{R}(\mathbf{s0})(\mathbf{R0s}^2)$. A term of the form $E(\mathbf{st})$ reduces then to $\mathbf{R0s}^2(E\mathbf{t})$. In the latter, the recurrence argument is a symbolic term $E\mathbf{t}$, representing a value

of the function we are in course of defining. Moreover, since that value is the iterative argument of the term $\mathbf{R0s}^2(E\mathbf{t})$, the meaningfulness of the definition hinges on admitting $E\mathbf{t}$ not only as some value, but as a natural number. Thus, E should be assumed as mapping $\mathbb{N}$ into $\mathbb{N}$ before its defining equations are admitted as meaningful. The same phenomenon is visible in the definition of Ackermann's function, there already in regard to function definition (whereas here it is manifested only in function computation).

Ramified recurrence, introduced in [15] as a sorted variant of recurrence, breaks this impredicativity by postulating a many-sorted data structure, with copies $\mathbb{A}_i$ ($i \geq 0$) of the algebra $\mathbb{A}$ in hand. The copy $\mathbb{A}_i$ is referred to as the *i'th tier*. Recurrence over tiers $\mathbb{A}_1 \ldots \mathbb{A}_i$ is then permitted only when the recurrence argument is in a tier $> i$. This prevents, in particular, that a recurrence argument refer back to the function being defined.

We convey ramified recurrence over $\mathbb{W}$ in a variant $\boldsymbol{\lambda}_1^*(\mathbb{W})$ of $\boldsymbol{\lambda}_1(\mathbb{W})$, obtained as follows. We refer to an unbounded list ι_p of base types, with ι_p intended to denote the tier $\mathbb{W}_p$. For each $p \geq 0$, and each constructor $\mathbf{c}$ (i.e., $\boldsymbol{\varepsilon}$, $\mathbf{0}$, or $\mathbf{1}$) we have a constant $\mathbf{c}_p$ denoting the copy of $\mathbf{c}$ in ι_p. (We drop the tier subscript when in no danger of confusion.) For each p we also have a branching operator $\mathbf{B}_p$, of type $\iota_p \to (\iota_p \to \iota_p)^2 \to \iota_0 \to \iota_p$. For each type τ which is a product of base types, we have a constant $\mathbf{R}_\tau$ of type $\tau \to (\tau \to \tau)^2 \to \iota_{p+1} \to \tau$, where p is the maximal tier in τ. The reductions of $\boldsymbol{\lambda}_1$ are augmented here with reductions for $\mathbf{B}_p$ and $\mathbf{R}_\tau$, similar to the reductions for $\mathbf{B}$ and $\mathbf{R}$ above, but subject to the revised types.

We claim that the constants $\mathbf{B}_p$ and $\mathbf{R}_\tau$ are as general as the tiered forms of branching and recurrence described in the preceding paragraph. The key observation is the definability of a downward coercion function D_p from $\mathbb{A}_{p+1}$ to $\mathbb{A}_p$: $D_p =_{\mathrm{df}} \mathbf{R}_{\iota_p}\boldsymbol{\varepsilon}_p\mathbf{0}_p\mathbf{1}_p$. The composition of these functions yields coercion functions $D_{p,q}$ from $\mathbb{A}_{p+1}$ to $\mathbb{A}_q$ for every $q \leq p$. Recurrence over τ can thus be driven by a recurrence argument of any type $p' > p$ by composing $D_{p',p+1}$ with $\mathbf{R}_\tau$.

The relevance of ramified recurrence to machine independent complexity is the following: [10]

Theorem **4** [16] *The functions over $\mathbb{A}$ definable in $\boldsymbol{\lambda}_1^*(\mathbb{A})$ are precisely the functions computable in polynomial time on a register machine over $\mathbb{A}$. In particular, the functions definable in $\boldsymbol{\lambda}_1^*(\mathbb{W})$ are precisely the functions computable in polynomial time on Turing machines, and the functions definable in $\boldsymbol{\lambda}_1^*(\mathbb{N})$ are the functions definable in linear space on Turing machine.* [11]

[10] A related result was proved earlier in [1]. However, the notion of "safe recursion" used there relies on a notion of "safe composition" which is not type-correct, does not conform to a reading of the tiers as base types, and therefore seems inappropriate for extensions to higher type computation.

[11] The latter is the second level $\mathcal{E}_2$ of the Grzegorczyk Hierarchy.

2.2 Ramified recurrence with type intersection: a motivating discussion

Theorem 4 and Cobham's Theorem [5] give two machine-independent characterizations of the functions over $\mathbb{W}$ computable in polynomial time. We wish to establish a correspondence between the extensions to higher type of these two approaches: the extension $\boldsymbol{\lambda}_1^-(\mathbb{W})$ of Cobham's system, and the higher type functionals definable in the ramified system $\boldsymbol{\lambda}_1^*(\mathbb{W})$.

Since in $\boldsymbol{\lambda}_1^-(\mathbb{W})$ one defines functionals over $\mathbb{W}$, whereas functionals definable in $\boldsymbol{\lambda}_1^*(\mathbb{W})$ are over over a *multi-sorted* data-structure, we must first clarify what we mean by definability in $\boldsymbol{\lambda}_1^*(\mathbb{W})$ of functionals over $\mathbb{W}$. Surely, we cannot admit all functionals with a ramified definition, because (as pointed out in the Introduction) the iteration functional would be definable, even though it is not definable in $\boldsymbol{\lambda}_1^-(\mathbb{W})$, and indeed maps doubling (a feasible function) to base-2 exponentiation (a non-feasible function). We thus restrict attention to functional definitions in $\boldsymbol{\lambda}_1^*(\mathbb{W})$ which are *tame* in a sense to be defined below.

In tandem with the restriction to tame terms, we will need an extension of the type system. The rationale is this. We wish to inductively transform each term M of $\boldsymbol{\lambda}_1^-(\mathbb{W})$ into a term M' of $\boldsymbol{\lambda}_1^*(\mathbb{W})$ which, tiers disregarded, defines the same functional as M. When $M \equiv M_0M_1$ we can stipulate $\boldsymbol{\lambda}_1^*(\mathbb{W})$-term M_0' and M_1', corresponding to M_0 and M_1, are already defined. However, if x, of type τ, is a free variable occurring in both M_0 and M_1, then x may be assigned in M_0' and M_1' two different tiered variants τ_0 and τ_1 of τ. This we resolve by simply adopting type intersection, and tentatively assigning to x in M the type $\tau_0 \cap \tau_1$. (Note that the intersection here is between tier-variants of the same type.) If x is not an argument of an application in M, then this simple measure resolves the issue. In general, however, there is a potential for typing conflict: we may have, for example, M_0 with a subterm $\mathbf{s}_0(x)$, where $\mathbf{s}_0 : \tau_0 \to \sigma_0$. Re-assigning to x the type $\tau_0 \cap \tau_1$ necessitates a modification of the tiering of $\mathbf{s}_0$ so as to yield a type $(\tau_0 \cap \tau_1) \to \sigma_0$. However, the tiering of functionals of higher rank has no computational consequences, since recurrence, where tiering matters, is restricted to functions over base types. Thus, the needed proliferation to higher type of type-intersection can go through harmlessly.

2.3 Generic tiering

We now define formally the extension $\boldsymbol{\lambda}_1^{\cap}(\mathbb{W})$ of $\boldsymbol{\lambda}_1^*(\mathbb{W})$. Given a ramified type τ, we write $\tilde{\tau}$ for the un-ramified type that arises from disregarding the tiers in τ. Call ramified types τ and σ *compatible* if $\tilde{\tau} = \tilde{\sigma}$. The formation rules of $\boldsymbol{\lambda}_1^*(\mathbb{W})$ for types are expanded as follows. If τ and σ are compatible types, then $\tau \cap \sigma$ is a type. We say that τ and σ are the *direct intersects* of $\tau \cap \sigma$; the relation "is an intersect of" is the reflexive and transitive closure of "is a direct intersect of". Constants and their types are precisely as in $\boldsymbol{\lambda}_1^*(\mathbb{W})$. If M is a term of $\boldsymbol{\lambda}_1^{\cap}(\mathbb{W})$, we write $\tilde{M}$ for the un-ramified form of M; note that since intersection is applied to compatible types only, $\tilde{M}$ is well-defined.

For correct typing of terms, we refer to the usual Curry-style derivations of typing statements.

2.4 Main result

We call a type $(\tau_1 \dots \tau_r) \to \sigma$ *critical* if it fails to consistently reduce tiers, in the following sense: σ is of the form $\iota_{p_1} \times \cdots \iota_{p_\ell}$ $(\ell \geq 1)$, and some τ_i is a product of base types, one of which is ι_q with $q \leq p_j$ for some j. A term of $\boldsymbol{\lambda}_1^{\cap}(\mathbb{W})$ is *tame* if no λ-abstracted variable therein has a type with a critical intersect.

Theorem **5** *A functional Φ over $\mathbb{W}$ is definable in $\boldsymbol{\lambda}_1^{-}(\mathbb{W})$ iff $\Phi = \tilde{\Psi}$ for some tiered-functional Ψ over $\mathbb{W}$ definable in $\boldsymbol{\lambda}_1^{\cap}(\mathbb{W})$ by a tame term.*

As mentioned in the Introduction, an alternative characterization of **BFF**, based on positive comprehension in the second order lambda calculus, is given in [17].

3 FROM RAMIFIED FUNCTIONALS TO BFF

Given functional identifiers (variables or constants) $\vec{\mathbf{f}}$, a *functional-polynomial in $\vec{\mathbf{f}}$* is a function $P : \mathbb{W}^r \to \mathbb{W}$ defined explicitly definable from $\vec{f}$, $\boldsymbol{\varepsilon}$, $\mathbf{0}$, $\mathbf{1}$, concatenation and word-multiplication. Here we take multiplication to mean $u * v =_{\mathrm{df}} 1^{|u|\cdot|v|}\boldsymbol{\varepsilon}$.[12]

A tiered functional Φ over $\mathbb{W}$ is *admissible* if it satisfies the following boundedness condition: If $\hat{\Phi} : (\tau_1 \dots \tau_\ell) \to \iota_p$, is explicitly defined from Φ and projections, then there is a functional-polynomial P such that $|\hat{\Phi}\vec{x}| \leq |P(\vec{y})| + \max_j[|z_j|]$, where $\vec{y}$ consists of the x_i's of higher type or of types with an intersect ι_q where $q > p$, and $\vec{z}$ consisting of the x_i's whose type is the intersection of types ι_p, $p \leq q$. Note that when $\vec{y}$ above is empty, P is a constant. Recall that the tiered first-order functions definable in $\boldsymbol{\lambda}_1^{*}(\mathbb{W})$ are admissible, by [16].

Since the collection of functional-polynomials is closed under application and under composition, we have:

Lemma **6** *The collection of admissible functionals is closed under application and under composition.* ⊣

Proposition **7** *Suppose a tiered-functional Φ over $\mathbb{W}$ is defined by a tame term $\lambda\vec{u}.\, M$. Let Φ' be a tiered-functional obtained by binding the critical-type arguments of Φ to admissible tiered-functionals $f_1 \dots f_m$. Then*

(i) *The functional Φ' is admissible.*

[12] This choice of a degenerated form of word-multiplication is motivated by the fact that we are only interested in the size of the output of functional-polynomials.

(ii) *If the un-tiered versions* $\tilde{f}_1 \dots \tilde{f}_m$, *of* $f_1 \dots f_m$ *respectively, are definable in* $\boldsymbol{\lambda}_1^-(\mathbb{W})$, *then so is the un-tiered version* $\widetilde{\Phi'}$ *of* Φ'.

Proof. By induction on M. The cases where M is one of the constructors $\boldsymbol{\varepsilon}$, $\mathbf{0}$ and $\mathbf{1}$, or one of the variables $\vec{u}$ are almost immediate. The cases where M is one of the constants $\mathbf{B}_p$ $(p \geq 0)$ are also straightforward.

Consider next the case $M \equiv \mathbf{R}_\tau$. Recall that τ must be a product of base types. Thus the critical-type arguments of $\mathbf{R}_\tau$ are the second and third (corresponding to the cases for the two successors). So Φ' here is a tiered-functional defined by $\lambda x_\epsilon w.\, \mathbf{R}_\tau x_\epsilon f_0 f_1 w$, where f_0, f_1 are admissible, i.e. for some constant a, $|f_i(z)| \leq a + |z|$. Hence $|\Phi'(x_\epsilon, w)| \leq |x_\epsilon| + |w| \cdot a$, and Φ' is admissible, with $P(x_\epsilon, w) =_{\rm df} x_\epsilon + w * \mathbf{1}^a \boldsymbol{\varepsilon}$ as bounding polynomial.

To demonstrate (2), assume that $\tilde{f}_0$ and $\tilde{f}_1$ are defined in $\boldsymbol{\lambda}_1^-(\mathbb{W})$, by F_0 and F_1 respectively. Let $J(x_\epsilon, w)$ be a term of $\boldsymbol{\lambda}_1^-(\mathbb{W})$ that defines P above (recall that a is a fixed value, depending on f_0 and f_1). Then $\lambda x_\epsilon, w.\bar{\mathbf{R}} x_\epsilon F_0 F_1 (J x_\epsilon w) w$ This concludes the induction's basis.

For the inductive step we consider λ-abstraction and application; we omit a discussion of pairing and projections, which are straightforward. Let $M = \lambda x^\tau . M_0$. Since M is assumed tame, the condition (1) for M is identical to (1) for M_0, which holds by IH. Also, (2) for M_0 trivially implies (2) for M.

Suppose $M = N^{\tau \to \sigma} Q^\tau$, and let Φ be the functional defined by $\lambda \vec{u}.M$. Let Φ_M be obtained by binding the critical-typed arguments of Φ to some fixed admissible functionals $\vec{f}$. Let Φ_n and Φ_Q be defined similarly for the functionals $\lambda \vec{u}.N$ and $\lambda \vec{u}.Q$. Towards showing that Φ_M is admissible, suppose that the functional $\hat{\Phi} : (\tau_1 \dots \tau_\ell) \to \iota_p$ is explicitly defined from Φ_M and projections. Then it is explicitly defined from Φ_N, Φ_Q and projections. The functionals Φ_N and Φ_Q are bounded by functional-polynomials, by IH. It follows, by a straightforward induction on (the length of) the definition of $\hat{\Phi}$ from Φ_N, Φ_Q, that $\hat{\Phi}$ too is bounded by a functional-polynomial.

Property (2) for M follows trivially from the IH for N and Q. ⊣

4 FROM BFF TO RAMIFIED FUNCTIONALS

When mapping $\boldsymbol{\lambda}_1^-(\mathbb{W})$ to $\boldsymbol{\lambda}_1^\cap(\mathbb{W})$ we use the boundedness condition on recurrence to enable appropriate tiering. The core of this mapping is given in the following Lemma.

Lemma **8** *For every* $p \geq r \geq 0$, *there is a* $\boldsymbol{\lambda}_1^*(\mathbb{W})$*-term* M *of type* $\iota_r \to \iota_{p+1} \to \iota_p$, *such that (the un-tiered variant of)* M *defines the function* $\restriction$.

Proof. Let $\tau = \iota_r \times \iota_p$. Define

$$N =_{\rm df} \pi_0 \mathbf{R}_\tau \langle \boldsymbol{\varepsilon}, u \rangle F_0 F_1 v$$

where

$$F_i = \lambda x.\, \langle \underline{case}\, (\pi_1 x, \pi_0 x, \mathbf{0}\pi_0 x, \mathbf{1}\pi_0 x),\ \mathbf{p}\pi_1 x \rangle.$$

By induction on its second argument we see that $\lambda u, v.\, N$ defines the $\restriction$ function reversed, i.e. $\lambda u, v.\, (\underline{rev}(u)) \restriction v$. Thus, the term

$$M =_{\mathrm{df}} \lambda u, v.\, \pi_0 \mathbf{R}_\tau \langle \boldsymbol{\varepsilon}, N \rangle F_0 F_1 v$$

defines $\restriction$. ⊣

It is easy to see that every function definable in $\boldsymbol{\lambda}_1^-(\mathbb{W})$ is already definable in $\boldsymbol{\lambda}_1^-(\mathbb{W})$ without product types. We are therefore free to focus on the product-free fragment in $\boldsymbol{\lambda}_1^-(\mathbb{W})$, somewhat simplifying technical details.

PROPOSITION **9** *Suppose Φ is a functional over $\mathbb{W}$, of product-free type $(\tau_1 \dots \tau_r) \to \iota$, defined in $\boldsymbol{\lambda}_1^-(\mathbb{W})$ by a normal term $M \equiv \lambda x_1^{\tau_1} \cdots x_r^{\tau_r}.M_0$. Then Φ is defined by some term $\underline{M}$ of $\boldsymbol{\lambda}_1(\mathbb{W})$ that has a tame typing in $\boldsymbol{\lambda}_1^{\cap}(\mathbb{W})$.*

Proof Outline. We prove, by induction on M_0, that there is a term $\underline{M}$ as above, with the property that for every $q \geq 0$ there is a $p \geq q$ such that for all $s \geq p$ $\underline{M}$ can be assigned in $\boldsymbol{\lambda}_1^{\cap}(\mathbb{W})$ the type $\vec{\sigma}^s \to \iota_q$. Here we write τ^s for the intersection of all possible non-critical ramified variants of τ, with tiers $\leq s$.

More precisely, our proof uses an auxiliary calculus, which results from augmenting $\boldsymbol{\lambda}_1^{\cap}(\mathbb{W})$ with the typing rule

$$\frac{\eta \vdash M : \iota_{p+i}}{\eta \vdash M : \iota_p}$$

That is, the role of the downward-coercion functions D_p is taken over by the typing rules. This permits an inductive proof in which we can ignore the need to insert D_p when called for. When done with the inductive proof, we can convert a typing derivation for a term M, which uses the coercion rule above, into a typing derivation of some variant of M that results from inserting instances of the functions D_p.

Turning to the induction basis of the proof of the claim above, the interesting case is the bounded recursion operator, i.e. M is $\overline{\mathbf{R}}$. Let

$$\underline{\overline{\mathbf{R}}} =_{\mathrm{df}} \lambda e, f_0, f_1, j, w.\mathbf{R} e f_0' f_1' w$$

where

$$f_i' \ \equiv\ \lambda v.((f_i(v) \restriction (jv)) \qquad (i = 0, 1)$$

Given q, we can type $\overline{\mathbf{R}}$ by setting, for any $r \leq q$, $e : \iota_q$, $v : \iota_q$, $f_i : \iota_q \to \iota_r$, $j : \iota_r \to \iota_{q+1} \to \iota_q$, $w : \iota_q + 1$. We can thus take $p = q + 1$.

For the induction step, consider first the case where M_0 ia an application. Since M_0 is assumed normal, it must then be of the form $AN_1 \cdots N_k$, where A is either a constant or a variable. The least trivial case with A a constant is $M_0 \equiv \overline{\mathbf{R}} E F_0 F_1 J W$ (i.e. $k = 5$). We let

$$\underline{M} =_{\mathrm{df}} \lambda x_1 \dots x_r.\mathbf{R}(\underline{E})(F_0')(F_1')(\underline{J})(\underline{W})$$

where

$$F'_i \equiv \lambda v.((\underline{F}_i(v) \restriction (\underline{J}v)) \qquad (i = 0, 1)$$

where $\underline{E}, \underline{F_0}, \underline{F_1}, \underline{J}$, and $\underline{W}$ are obtained by IH applied to E, F_0, F_1, J, and W, respectively. Next, suppose given $q \geq 0$. Towards defining the appropriate value for p, consider the variables $x_1 : \tau_1, \ \ldots \ , x_r : \tau_r$ free in M_0, and let p_E and p_J be obtained for the given q by IH applied to E and J, respectively; let p_N be obtained for $q{+}1$ by IH applied to N; and let p_0 and p_1 be obtained for the value $q = 0$ by IH applied to F_0 and F_1. If we set $p =_{\text{df}} \max[p_E, p_0, p_1, p_J, p_N]$, and assign $x_i : \tau_i^p$, then, using the fundamental rules for type intersection, we obtain as derived typings $\underline{E} : \iota_q$, $\underline{F}_i : \iota_q \to \iota_0$, $\underline{J} : \iota_0 \to \iota_{q+1} \to \iota_q$, and $\underline{W} : \iota_q + 1$, so p satisfies the required property with respect to $\underline{M}$.

The other cases for $\overline{\mathbf{R}}$ (i.e. fewer than 5 arguments) are included in the above by η-conversion. The cases where A is one of the remaining constants, $\boldsymbol{\varepsilon}$, $\mathbf{0}$, $\mathbf{1}$, $\mathbf{B}_p$ or $\restriction$ are straightforward.

If A above is a variable, x_1 say, we let

$$\underline{M} =_{\text{df}} \lambda x_1^{\tau_1} \ldots x_r^{\tau_r}.\, x_1 \underline{N}_1 \cdots \underline{N}_k$$

where $\underline{N}_i$ is obtained by IH applied to N_i $(i = 1 \ldots k)$. Let σ_i be the (untiered) type of N_i Given $q \geq 0$, let p_i be obtained by IH for q and N_i. Let $\bar{p} =_{\text{df}} \max[p_1 \ldots p_k]$. Then all tiers in $\sigma_i^{\bar{p}}$ are $\leq p$. Choosing $p =_{\text{df}} \bar{p}{+}1$, we guarantee that the type

$$\sigma_1^{\bar{p}} \to \cdots \to \sigma_1^{\bar{p}} \to \iota_q$$

is one of the intersects of τ_1^p, and therefore the typing is correct. This concludes the proof of the induction step for an applicative M_0.

The case of λ-abstraction is trivial. $\dashv$

References

[1] Stephen Bellantoni and Stephen Cook. A new recursion-theoretic characterization of the poly-time functions, 1992.

[2] Samuel Buss. *Bounded Arithmetic*. Bibliopolis, Naples, 1986.

[3] Samuel Buss. The polynomial hierarchy and intuitionistic bounded arithmetic. In *Structure in Complexity*, LNCS 233, pages 77–103, Berlin, 1986. Springer-Verlag.

[4] Peter Clote. A note on the relation between polynomial time functionals and constable's class k. In Hans Kleine-Büning, editor, *Computer Science Logic*, LNCS 1092, pages 145–160, Berlin, 1996. Springer-Verlag.

[5] A. Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel, editor, *Proceedings of the International Conference on Logic, Methodology, and Philosophy of Science*, pages 24–30. North-Holland, Amsterdam, 1962.

[6] Robert Constable. Type 2 computational complexity. In *Fifth Annual ACM Symposium on Theory of Computing*, pages 108–121, New York, 1973. ACM.

[7] Stephen Cook. Computability and complexity of higher type functions. In Y. Moschovakis, editor, *Logic from Computer Science*, pages 51–72. Springer-Verlag, New York, 1991.

[8] Stephen A. Cook and Alasdair Urquhart. Functional interpretations of feasible constructive arithemtic (extended abstract). In *Proceedings of the 21st ACM Symposium on Theory of Computing*, pages 107–112, 1989.

[9] Stephen A. Cook and Alasdair Urquhart. Functional interpretations of feasible constructive arithemtic. *Annals of Pure and Applied Logic*, 63:103–200, 1993.

[10] Kurt Gödel. Über eine bisher noch nicht benutzte erweiterung des finiten standpunktes. *Dialectica*, 12:280–287, 1958.

[11] A. Grzegorczyk. Some classes of recursive functions. In *Rozprawy Mate. IV*. Warsaw, 1953.

[12] R. Irwin, B.M. Kapron, and J. Royer. On characterizations of the basic feasible functionals part i. *Journal of Functional Programming*, 11:117–153, 2001.

[13] B.M. Kapron and S.A. Cook. A new characerization of type-2 feasibility. *SIAM Journal of Computing*, 25:117–132, 1996.

[14] Bruce Kapron and Stephen Cook. Characterizations of the basic feasible functionals of finite type. In S. Buss and P. Scott, editors, *Feasible Mathematics*, pages 71–95. Birkhauser-Boston, 1990.

[15] Daniel Leivant. Subrecursion and lambda representation over free algebras. In Samuel Buss and Philip Scott, editors, *Feasible Mathematics*, Perspectives in Computer Science, pages 281–291. Birkhauser-Boston, New York, 1990.

[16] Daniel Leivant. Ramified recurrence and computational complexity I: Word recurrence and poly-time. In Peter Clote and Jeffrey Remmel, editors, *Feasible Mathematics II*, Perspectives in Computer Science, pages 320–343. Birkhauser-Boston, New York, 1994.

[17] Daniel Leivant. Implicit computational complexity for higher type functionals. In Julian Bradfield, editor, *Computer Science Logic 2002*, 2002.

[18] Kurt Mehlhorn. Polynomial and abstract subrecursive classes. *JCSS*, 12:147–178, 1976.

[19] Anil Seth. Some desirable conditions for feasible functionals of type 2. In *Proceedings, Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 320–331, Washington, DC, 1993. IEEE Computer Society Press.